VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

# Benchmarking API pro Bluetooth komunikaci pro multiplatformní webové aplikace

Benchmarking of Bluetooth Communication APIs for Multi-Platform Web Applications

Radovan Rečka

Bakalářská práce

Vedoucí práce: Ing. Pavel Moravec, Ph.D.

Ostrava, 2021

## Abstrakt

Práce se zabývá testováním knihoven pro Bluetooth komunikaci. Skládá se ze dvou částí, kde první část je spíše teoretická. Je v ní popsána technologie Bluetooth a její dělení na Bluetooth Classic a Bluetooth Low Energy. Práce rovněž rozebírá, jaké knihovny pro Bluetooth komunikaci využít v závislosti na tom, zda budeme chtít vytvořit webovou, progresivní, nebo hybridní aplikaci. Práce čtenáře obeznámí s Web Bluetooth API a dalšími Apache Cordova pluginy. Závěr teoretické části je věnován návrhu testů a popisu možností měření jejich trvání.

V praktická část se zaměřujeme na implementaci kódu pro Bluetooth komunikaci prostřednictvím Bluetooth knihoven. Práce provádí čtenáře implementací jednotlivých fází komunikace s vybranými knihovnami a jejich testováním. V druhé polovině praktické části je popsáno testovací prostředí a jsou vyhodnoceny výsledky provedených testů.

## Klíčová slova

Benchmark, Bluetooth API, Web Bluetooth API, Apache Cordova, plugin, Bluetooth Low Energy, Bluetooth Classic

## Abstract

The thesis deals with testing frameworks for Bluetooth communication. It consists of two parts, where the first part is rather theoretical. It describes Bluetooth technology and its division into Bluetooth Classic and Bluetooth Low Energy. Further, it discuss which frameworks for Bluetooth communication to use depending on whether we want to create a web, progressive or hybrid application. The work will introduce the reader to the Web Bluetooth API and other Apache Cordova plugins. The conclusion of the theoretical part is concentrating on the design of tests and a description of the possibilities of measuring their duration.

In the practical part we focus on the implementation of code for Bluetooth communication via Bluetooth frameworks. Work takes the reader through the process of implementation the various phases of communication with selected frameworks and their testing. The second half of the practical part describes the test environment and evaluates the results of the tests.

## Keywords

Benchmark, Bluetooth API, Web Bluetooth API, Apache Cordova, plugin, Bluetooth Low Energy, Bluetooth Classic

## Poděkování

Chtěl bych poděkovat svému vedoucímu bakalářské práce Ing. Pavlu Moravcovi, Ph.D. za odborné vedení, rady, věcné připomínky, vstřícnost a časovou flexibilitu při konzultacích. Vždy se mi v průběhu zpracování bakalářské práce s ochotou a trpělivostí věnoval.

# Obsah

# Seznam použitých zkratek a symbolů

| | | |
|---|---|---|
| API | – | Application Programming Interface |
| SIG | – | Special Interest Group |
| UUID | – | Universally unique identifier |
| SPP | – | Serial Port Profile |
| A2DP | – | Advanced Audio Distribution Profile |
| AVRCP | – | Audio/Video Remote Control Profile |
| BPP | – | Basic Printing Profile |
| FTP | – | File Transfer Profile |
| GATT | – | Generic Attribute Profile |
| HFP | – | Hands-Free Profile |
| HID | – | Human Interface Device Profil |
| SDAP | – | Service Discovery Application Profile |
| SDP | – | Service Discovery Procedure |
| GAP | – | Generic Access Profile |
| HTML | – | Hypertext Markup Language |
| CSS | – | Cascading Style Sheets |
| PWA | – | Progressive Web Apps |
| GPS | – | Global Positioning System |
| BLE | – | Bluetooth Low Energy |
| LE | – | Low Energy |
| MAC | – | Media Access Control |
| SSD | – | Solid-state Drive |
| MIUI | – | Mi User Interface, vyslovováno Me You I |
| iOS | – | iPhone Operating System |

# Seznam obrázků

# Kapitola 1

# Úvod

V bakalářské práci se zabýváme benchmarkingem nejvíce užívaných multiplatformních Bluetooth API. Technologii Bluetooth jsem se rozhodl věnovat, protože mi přijde zajímavá a chtěl jsem si vyzkoušet práci s ní. Zároveň se mi zamlouvá myšlenka, že můžeme přistupovat k Bluetooth bez závislosti na platformě.

Technologie Bluetooth a později Bluetooth Smart od svého vzniku prošly postupně významným vývojem. Jak samotná technologie Bluetooth, tak i její nástroje pro vývojáře, se postupem doby zlepšovaly a optimalizovaly do podoby, v jaké jsou dnes používány.

Existuje spousta Bluetooth API, pomocí nichž jsme schopni vytvářet programy téměř pro jakákoliv zařízení, která disponují technologií Bluetooth. Občas si ani neuvědomujeme, že věci každodenního užití pracují s jmenovanými technologiemi. Zároveň je třeba poukázat, že neexistuje jednotný způsob, jak k Bluetooth přistupovat. Vše je dáno tím, že existuje velké množství Bluetooth API, z kterých si vývojář může při řešení projektu vybrat.

Právě s důrazem na různorodost technologie Bluetooth se s ní na začátek v první kapitole seznámíme. V uvedené části práce se zaměříme i na to, jak se technologie Bluetooth dělí a jaké Bluetooth profily jsou nejužívanější a k čemu jednotlivé profily slouží.

V další kapitole budeme hledat způsob, jak můžeme vytvořit aplikaci, která komunikuje skrze Bluetooth a které skutečnosti je nutné zohlednit při vytváření projektu řešícího komunikaci s použitím uvedené technologie. Řekneme si zde i něco o standardu ECMAScript. Poté se seznámíme se zařízeními, se kterými budeme komunikovat v našich benchmarcích. Následně přejdeme k hlavnímu bodu této kapitoly, nímž je popis rozdílů mezi webovou aplikací, hybridní aplikací v Apache Cordova a progresivní webovou aplikací.

Ve čtvrté kapitole provedeme rešerši nejpoužívanějších Bluetooth knihoven. U každé knihovny si popíšeme, pro které platformy ji můžeme využít, mezi jakými zařízeními nám umožní komunikovat, jaké jsou požadavky na jejich užívání, ale i jaká platí omezení v jejich využití a na co si dát u jednotlivých knihoven pozor.

Pátá kapitola nás seznámí s návrhem benchmarkingových testů. Dozvíme se zde informace o Performance API a také, jak můžeme získávat hodnoty z testů. Následně si řekneme, jak testy budou vypadat a co bude jejich obsahem.

Další kapitola nás obeznámí s implementací testů navržených API. Dále se zde dozvíme, jak jsou naimplementovány benchmarking testy. Uvedu zde i svůj subjektivní názor na zkušenosti s prací s danými API.

Předposlední kapitola nás seznámí se zařízeními, na kterých budou prováděny testy prostřednictvím vypracovaného kódu. Následně si testy vyhodnotíme a výsledky zhodnotíme.

V závěru shrneme získané výsledky a určíme, které z testovaných Bluetooth API je pro naše účely prostřednictvím Bluetooth nejlepší.

# Kapitola 2

# Seznámení se s technologií Bluetooth

Technologie Bluetooth je velmi oblíbenou technologií pro bezdrátový přenos. V současnosti se o technologii stará Bluetooth Special Interst Group (užívá se zkratky Bluetooth SIG). Jmenovaná organizace zodpovídá za vývoj standardů technologie Bluetooth a jejich licencování výrobcům. Technologie Bluetooth během svého vývoje prošla mnoha vylepšeními a měla několik verzí. Stále menší podíl uživatelů používá verzi 4.0. V tomto případě jde spíše o využívání starších mobilních zařízení, která tímhle standardem disponují. Aktuálně vyráběná zařízení jsou podporována standardem 4.2. Největší firmy do svých vlajkových zařízení již aplikují standard 5.x, který je nejnovější. Během vývoje standardů se stále zlepšovaly klíčové parametry Bluetooth technologie. Jednalo se o rychlost přenosu dat, vzdálenost, na kterou můžeme data přenést a o energetickou náročnost. Podle těchto klíčových parametrů se nám technologie Bluetooth rozděluje na dvě základní verze. První verzí je Bluetooth Low Energy a druhou Bluetooth Classic [1].

Úvodem je potřebné sdělit, co znamená UUID a charakteristika. Význam UUID je univerzální unikátní identifikátor. Používá se buď jako 16ti, 32 nebo 128mi bitový řetězec. Každá služba na zařízení má své vlastní UUID. Stejně je tomu i u charakteristik. Charakteristiky jsou položky dat, které se vztahují ke konkrétnímu vnitřnímu stavu zařízení nebo k určitému stavu prostředí. Mají typ, hodnotu, různé vlastnosti a některá oprávnění [2].

## 2.1 Bluetooth Classic

Bluetooth Classic je rovněž navržena pro práci s malým využitím energie. Na rozdíl od Bluetooth Low Energy přenos probíhá přes 79 kanálů a rychlost přenosu dat může dosahovat od 1 Mb/s do 3 Mb/s [1]. Ke komunikaci některé periferie využívají SPP profil a jiné A2DP profil. Bluetooth Classic je vhodnější užívat v momentě, kdy zařízení komunikují na malou vzdálenost. Dobrým příkladem užití je Handsfree. Obě zařízení nejsou od sebe příliš vzdálená. Pro lepší kvalitu hovoru je nutná vyšší přenosovou rychlost a více kanálů pro přenos.

## 2.2 Bluetooth Low Energy

Bluetooth Low Energy, také nazýváme Bluetooth Smart, je navržena pro provoz s nízkou energetickou náročností. Přenos probíhá přes 40 kanálů a podporuje rychlost přenosu od 125 Kb/s do 2 Mb/s. Výkon Bluetooth Low Energy se pohybuje od 1 mW do 100 mW. Bluetooth Low Energy podporuje více síťových topologií včetně point-to-point, broadcast a mesh sítě [1]. Ke komunikaci využívá GATT profil. Poprvé se Bluetooth Low Energy objevila u Bluetooth verze 4.0. Toto řešení napomohlo rozvoji nositelných zařízení, která využívají dané technologie. Další možností bylo jednodušší nasazení přenosných senzorů, které se díky tomuto vylepšení daly využívat po delší dobu.

## 2.3 Nejužívanější Bluetooth profily

Aby Bluetooth správně komunikoval se všemi zařízeními, využívá řadu definovaných profilů. To znamená, že pro určitý use case můžeme využít optimálně jen jeden profil. Může se stát, že některé use casy budou využívat více profilů.

### 2.3.1 Audio/Video Remote Control Profile (AVRCP)

Uvedený profil je zodpovědný za vzdálené ovládání Bluetooth zařízení. Většinou jde o zařízení typu: televize, sluchátka, reproduktory, smartphony a různá jiná další.

Profil přenosu definuje dvě role: Controller a Target [3]. Controllerem je obvykle dálkové ovládání a Targetem je zařízení, které chceme ovládat.

### 2.3.2 Basic Printing Profile (BPP)

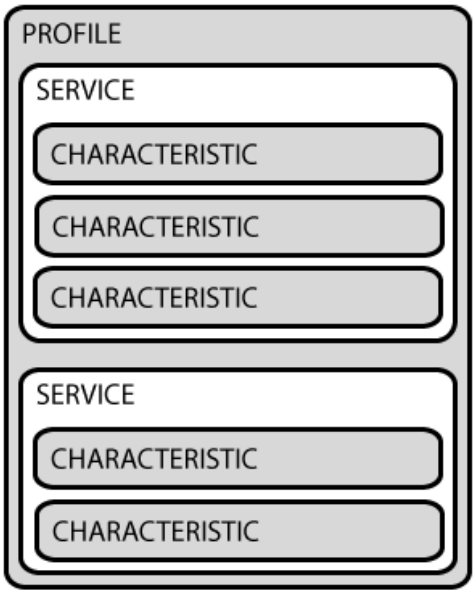
BPP profil nám umožňuje odesílat data, např.: textové zprávy, e-maily, obrázky a jiné do tiskáren na základě tiskových úloh. Tiskový profil definuje dvě role. První rolí je Tiskárna, tedy zařízení, které má provést tisk.

Druhou rolí je Odesílatel. Zde se jedná o zařízení, které chce tisknout data [4]. Výhodou uvedeného profilu na rozdíl od jiných profilů, které jsou určené k tisku, je, že nepotřebuje žádné ovladače pro specifickou tiskárnu.

### 2.3.3 File Transfer Profile (FTP)

Jak už nám název napovídá, jedná se o profil, který používáme k bezdrátovému přenosu souborů mezi dvěma zařízeními. Opět jsou zde definovány dvě role. První zařízení vystupuje v roli Klienta. Jedná se o zařízení, které iniciuje operaci posílání nebo stažení souborů ze serveru.

Druhé zařízení vystupuje v roli serveru. Server je cílové zařízení, do kterého posíláme nebo z kterého stahujeme soubor [3].




Obrázek 2.1: Vnořené objekty u Generic Attribute Profilu, převzato z [5]

### 2.3.4 Generic Attribute Profile (GATT)

U profilu GATT se data přenáší pomocí Services a Characteristics. K ukládání Services, Characteristics a dat s nimi spojenými používá jednoduchou tabulku 16-ti bitových ID. Nejedná se o podobné řešení jako u jiných technologií nebo profilů. Je to z důvodu, že profil v popisovaném případě není na periferním zařízení, ale je to předdefinovaná kolekce služeb. Příkladem může být profil pro sledování srdeční frekvence. Ten kombinuje Heart Rate Service a Device Information Service [5]. Můžeme vidět (viz Obrázek 2.1), jak v GATT profilu jsou vnořené služby a v nich jednotlivé charakteristiky.

### 2.3.5 Hands-Free Profile (HFP)

HFP profil používáme na úspěšné spojení telefonních hovorů pomocí Bluetooth zařízení. V dnešní době jde spárovat Handsfree jak s mobilním telefonem, tak i s autem samotným. Jsou zde dvě role. První rolí je Audio Gateway. Audio Gateway obvykle reprezentuje mobilní telefon nebo systém auta.

Druhou rolí je Hands-Free Unit [3], která je reprezentovaná samotným handsfree zařízením a funguje tedy jako výstup zvuku.

### 2.3.6 Human Interface Device Profile (HID)

Human Interface Device profil popisuje protokoly a funkce, které mají používat zařízení, jako jsou bezdrátové klávesnice, myši, herní kontrolery a další bezdrátová vybavení, která jsou ovládána člověkem. Opět zde rozlišujeme dvě role. První rolí je Human Interface Device. Jde o zařízení, které poskytuje data od a pro člověka [4] .

Druhou rolí je Host. Jedná se o zařízení, které používá služby a zpracovává je dále.

### 2.3.7 Service Discovery Application Profile (SDAP)

Profil nám popisuje, jak by aplikace měly zjišťovat služby na vzdáleném zařízení pomocí tzv. Service Discovery Procedure (SDP). SDAP se postupně dotazuje, skenuje a zjišťuje jaké služby může použít pomocí SDP [4]. Mezi jednu z jeho dvou rolí patří Local Device. Zařízení inicializuje celou komunikaci.

Druhá je role Remote Device. Může jich být i více. Odpovídají na požadavky služeb z Local Device.

### 2.3.8 Serial Port Profile (SPP)

SPP profil nám říká, jak máme nastavit virtuální sériové porty a jak můžeme připojit dvě Bluetooth zařízení. Zde se role jmenují jednoduše: Zařízení A a Zařízení B [4], kde Zařízení A iniciuje navázání spojení a Zařízení B akceptuje spojení.

### 2.3.9 Generic Access Profile (GAP)

Generic Access Profile nám udává, jak bude naše zařízení viditelné pro okolní svět, a jak se mohou, popřípadě nemohou, dvě zařízení interagovat navzájem. Je zde definováno více rolí, ale udávají se dvě hlavní role. První rolí je centrální zařízení. Toto zařízení obvykle představuje mobilní telefon nebo tablet, ke kterému se připojujeme.

Druhým zařízením je peripheral device, nebo česky periferní zařízení. Jde o malé zařízení s malým využitím energie. K těmto zařízením se obvykle připojujeme [6].

# Kapitola 3

# Tvorba mobilních aplikací, komunikujících skrze Bluetooth

Mobilní aplikace, které komunikují skrze Bluetooth, mohou být tří druhů, podle toho, jak byly vytvořeny. První možnost, jak vytvořit aplikaci pro mobilní zařízení, je naprogramování webové aplikace. Zde můžeme aplikaci vytvořit pomocí značkovacího jazyka HTML (v dnešní době se používá verze HTML5), kaskádových stylů CSS a multiplatformního skriptovacího jazyka JavaScript.

JavaScript byl vytvořen Brendanem Eichem v roce 1995 a stal se ECMA standardem v roce 1997 (ECMA je mezinárodní soukromá nevýdělečná organizace pro normalizaci informačních a komunikačních systémů). Od roku 1997 postupně vznikaly verze ECMAScriptu, které se označují ES a které přináší nové funkce do jazyka. Navázaly verze ECMAScript 1 - ECMAScript 6. Od roku 2016 se nové verze jmenují podle roků. Vznikly verze ECMAScript 2016 - ECMAScript 2018 [7]. ECMAScript 1 - 6 je podporován ve všech moderních prohlížečích.

Dále můžeme vytvořit hybridní aplikaci v Apache Cordova. Posledním řešením je tzv. progresivní webová aplikace.

Při programování mobilních aplikací, které komunikují přes Bluetooth, je potřebné vědět, s jakými zařízeními budou komunikovat. Pro určitá zařízení můžeme naprogramovat kód, abychom při vyhledávání zařízení nalezli jen ta zařízení, která disponují námi zvolenou službou, popřípadě charakteristikou. Pro bakalářskou práci jsem si vybral zařízení TI SensorTag CC2650STK od společnosti Texas Instruments (viz Obrázek 3.1). Periferní zařízení mě zaujalo nabídkou velkého množství senzorů. Jednalo se o senzor okolního světla, magnetický senzor, senzor vlhkosti, senzor tlaku, akcelerometr, gyroskop, senzor teploty objektu a okolí a jiné. Hodnotou, kterou budeme jeho prostřednictvím získávat během testů, bude teplota. Snadno můžeme provést i kontrolu získaných hodnot prostřednictvím jiných měřidel teploty [8].

Zařízení je malé a lehce přenosné.

K zařízení je sepsán v elektronické podobě vcelku dobrý manuál, který uživateli poskytuje dostatek informací. Negativně jsem vnímal špatnou přehlednost písemné dokumentace. Někdy bylo

Obrázek 3.1: TI SensorTag CC2650STK, převzato z [9]

potřebné si údaje o přístroji dohledat z jiných zdrojů.

Na obrázku 3.2 můžeme vidět přehled UUID služeb, kterými zařízení disponuje. Pro nás jsou důležité hodnoty pod nadpisem Temperature. V ukázkách kódů můžeme vidět i AA80, což je UUID služby akcelerometru, toto UUID jsme v příkladech zvolili při vyhledávání, abychom nalezli zařízení SensorTag CC2650STK, a ne jeho starší variantu s jinými vlastnostmi senzoru teploty.

Rozeznáváme několik základních druhů mobilních aplikací, které jsou schopny komunikovat přes Bluetooth.

## 3.1 Webová aplikace

Jak už název napovídá webová aplikace se spouští přes webový prohlížeč. Její uložení může být jak lokální, tak na serveru. Co se týče přístupu k Bluetooth jsou webové API (konkrétně Web Bluetooth API), na experimentální úrovni. To znamená, že běžní uživatelé mohou mít problém s využitím určitých funkcí. Odborníci, pracující se zařízeními na vyšší, než jen uživatelské úrovni, mohou být schopni zpřístupnit si dané funkce sami. Pro méně technicky nadané uživatele se může jednat o problém. Proto se doporučuje využívat metod, které nevyžadují žádné dodatečné povolení. Jako u každé jiné aplikace i zde jsou výhody a nevýhody.

**Výhody:**

- Nezávislost na platformě a konkrétním zařízení
- Minimální nároky na počítač
- Aktuálnost aplikace (pokud se jedná o aplikaci na serveru)
- Lehké napojení na další funkce/aplikace v síti

**Nevýhody:**

| | UUID | Function |
|---|---|---|
| Battery | | |
| Service | 0x180F | Battery service |
| Level | 0x2A19 | Battery level |
| Device Information | | |
| Service | 0x180A | Device information service |
| Manufacturer name string | 0x2A29 | Manufacturer name |
| Model number string | 0x2A24 | Model number |
| Firmware revision string | 0x2A26 | Firmware revision |
| System ID | 0x2A23 | System ID |
| Current Time Client | | |
| Service | 0x1805 | Current time client service |
| Current time characteristics | 0x2A2B | Get current time from server |
| Button Less DFU | | |
| Service | 0xFE59 | Button less DFU service |
| DFU | 8ec90003-f315-4f60-9fb8-838830daea50 | DFU |
| Temperature | | |
| Service | AA00 | Temperature service |
| Data | AA01 | Real time sensor data (see table 3) |
| Config | AA02 | Set/get sensor on/off state |
| Period | AA03 | Set/get reporting period |
| Humidity | | |
| Service | AA20 | Humidity service |
| Data | AA21 | Real time sensor data (see table 3) |
| Config | AA22 | Set/get sensor on/off state |
| Period | AA23 | Set/get reporting period |
| Luxometer (for 8130-A only) | | |
| Service | AA70 | Luxometer service |
| Data | AA71 | Real time sensor data (see table 3) |
| Config | AA72 | Set/get sensor on/off state |
| Period | AA73 | Set/get reporting period |
| Accelerometer (for 8130-A only) | | |
| Service | AA80 | Accelerometer service |
| Data | AA81 | Real time sensor data (see table 3) |
| Config | AA82 | Set/get sensor on/off state |
| Period | AA83 | Set/get reporting period |
| IO | | |
| Service | AA64 | IO service |
| Data | AA65 | IO control (see table 2) |

Obrázek 3.2: Přehled UUID hodnot pro zařízení SensorTag, převzato z [10]

- Při umístění na serveru je potřeba síťové připojení
- Povolování nestandartních funkcí
- Různé prohlížeče nemusí být kompatibilní

## 3.2 Hybridní aplikace v Apache Cordova

Hybridní aplikace v Apache Cordova jsou aplikace, které nám spojují nativní technologie a webové technologie. To znamená, že pomocí specializovaných nástrojů a webových technologií můžeme vytvořit po následné konverzi aplikace pro jednotlivé platformy [11]. U Apache Cordova k tomu využíváme pluginy, které importujeme do projektu a následně použijeme metody, které se nám zpřístupnily. Jsou psány pomocí webových technologií (HTML, CSS a Javascript) a následně místo toho, aby se aplikace zobrazila v prohlížeči uživatele, spustí se ve výchozím prohlížeči a tváří se jako nativní aplikace. I zde máme výhody a nevýhody.

**Výhody:**

- Snadnější údržba (díky využití webových technologií)
- Snadnější přidávání nových funkcí
- Možnost využívat funkce zařízení (nativní API jsou k dispozici)
- Možnost propojení s webovými službami
- Může pracovat off-line v závislosti na funkcích aplikace

**Nevýhody:**

- Složitější aplikace s mnoha funkcemi budou pomalejší
- Nedá se ovlivnit zabezpečení systémového prohlížeče [12]

## 3.3 Progresivní webová aplikace

Progresivní webové aplikace se označují PWA. Progresivní webové aplikace jsou novinkou, ale v poslední době poměrně rozšířené. Jedná se o webové aplikace, které se snaží vypadat a chovat jako nativní aplikace. Znamená to, že PWA lze instalovat a přistupovat k nim na mobilních zařízeních (pracují off-line a mohou posílat notifikace). Mohou také používat hardwarové funkce jako jsou fotoaparát a GPS, a v našem případě Bluetooth. Na každou platformu jsou aplikace optimalizovány, a proto jsou jejich funkce a načítání rychlejší. Mezi výhody a nevýhody patří následující.

**Výhody:**

- Vysoká rychlost načítání
- Práce off-line
- Rychlejší a levnější vývoj než u nativních aplikací
- Dobře se adaptuje na zařízení
- Uživatelská interakce se podobá nativním aplikacím
- Instalace z Google Play
- Přidání ikony na plochu

**Nevýhody:**

- Omezení ze strany hardwaru a operačního systému co se týče funkcí
- Slabší podpora pro zařízení na platformě iOS
- Vyšší spotřeba energie [12]

# Kapitola 4

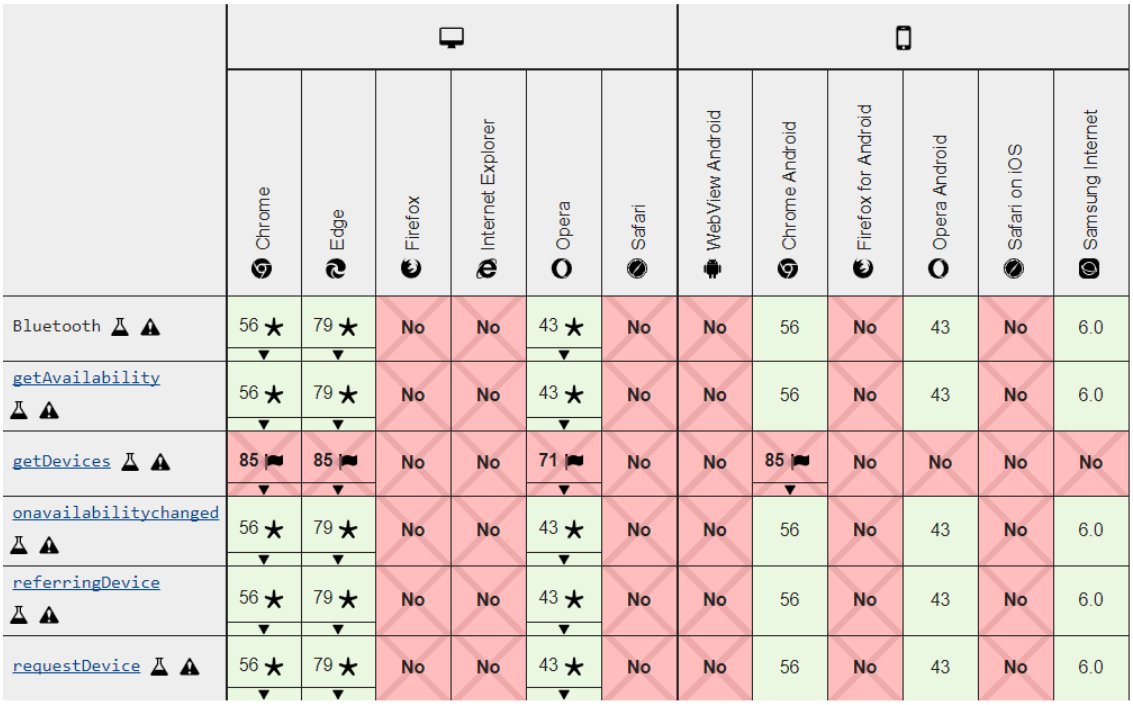
# Nejpoužívanější Bluetooth knihovny

Abychom mohli využívat Bluetooth, máme několik možností, jak k němu přistupovat. Pro účely této práce můžeme provést pomyslné rozdělení na experimentální Web Bluetooth a na Apache Cordova pluginy. Jak u experimentálního Web Bluetooth, tak u Apache Cordova pluginů je potřebné ověřit, jaké platformy, popřípadě prohlížeče jsou podporovány a jaká omezení pro jednotlivé knihovny platí. Dá se předpokládat, že minimální požadavky na verzi operačního systému Android a iOS vyplývají z úvodní podpory prostřednictvím Bluetooth Low Energy v Androidu a iOS.

Prvním krokem bylo prostudování nejvíce užívaných API pro Bluetooth komunikaci a Web Bluetooth API. K tomu jsem použil oficiální stránky Apache Cordova, které naleznou správný plugin pro práci s nativními funkcemi zařízení. Na již zmíněném webu jsem prošel všechny pluginy, abych nalezl ty nejvíce používané. U každého pluginu se mimo jiné sledují týdenní stažení. Díky tomu bylo jednoduché zjistit, že pluginy jsou aktuálně stále užívané a nejsou zastaralé.

## 4.1 Web Bluetooth API

Jedno z nejpoužívanějších API pro Bluetooth je experimentální Web Bluetooth API. Jmenované API nám pomáhá připojit se a interagovat s Bluetooth periferiemi. Jde o velmi populární možnost, jak přistupovat k Bluetooth zařízením. Kompatibilita s prohlížeči je dostatečná. Mezi prohlížeče, které API podporují na počítačích, patří Google Chrome, Microsoft Edge a Opera. U mobilních zařízení je podpora u Google Chrome pro Android, Opera pro Android a Samsung Internet (viz Obrázek 4.1) Je nutné pracovat se skutečností, že se jedná o experimentální Bluetooth API, proto mohou nastávat problémy s tím, zda má uživatel povoleny určité funkce (zejména *getDevices*, viz Obrázek 4.1) [13]. Pro zpřístupnění těchto funkcí je potřeba povolit nová oprávnění v `chrome://flags`. Nicméně se dané funkce nedoporučuje používat, mohou se v průběhu doby měnit.



| | Chrome | Edge | Firefox | Internet Explorer | Opera | Safari | WebView Android | Chrome Android | Firefox for Android | Opera Android | Safari on iOS | Samsung Internet |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Bluetooth | 56 ★ | 79 ★ | No | No | 43 ★ | No | No | 56 | No | 43 | No | 6.0 |
| getAvailability | 56 ★ | 79 ★ | No | No | 43 ★ | No | No | 56 | No | 43 | No | 6.0 |
| getDevices | 85 | 85 | No | No | 71 | No | No | 85 | No | No | No | No |
| onavailabilitychanged | 56 ★ | 79 ★ | No | No | 43 ★ | No | No | 56 | No | 43 | No | 6.0 |
| referringDevice | 56 ★ | 79 ★ | No | No | 43 ★ | No | No | 56 | No | 43 | No | 6.0 |
| requestDevice | 56 ★ | 79 ★ | No | No | 43 ★ | No | No | 56 | No | 43 | No | 6.0 |

Obrázek 4.1: Podpora Web Bluetooth v prohlížečích, převzato z [14]

## 4.2 Cordova-plugin-bluetooth-serial

Tento plugin nám dovoluje sériovou komunikaci přes Bluetooth a byl vytvořen pro komunikaci mezi zařízeními Android, iOS a Arduino.

**Podporované platformy:**

- Android
- iOS RedBearLab BLE hardware, který slouží k rozšíření hardwaru zařízení s operačním systémem iOS
- Windows Phone 8
- Prohlížeč

**Omezení:**

- Telefon musí zahájit Bluetooth připojení
- iOS Bluetooth LE vyžaduje iPhone 4S, iPhone 5, iPod 5 nebo iPad 3+, dá se očekávat, že podporována budou i novější zařízení
- Může být problém s propojením dvou Android zařízení a dvou iOS zařízení [15]

## 4.3 Cordova-plugin-bluetooth-serial2

U tohoto pluginu se jedná o starší verzi předchozího pluginu. Z důvodu kratšího období podpory je více zastaralý. Týdenní stažení tohoto pluginu dokládá, že jeho předchozí verze je oblíbenější. Výše zmíněný plugin už není aktualizován [16].

## 4.4 Cordova-plugin-bluetoothClassic-serial

Poslední plugin, který využívá sériovou komunikaci přes Bluetooth. Jde o poslední jmenovaný plugin, který vychází z našeho prvního pluginu, ovšem můžeme zde najít několik zásadních změn. Plugin je napsán pomocí iOS Accessory Frameworku (MFi) pro podporu Classic Bluetooth v iOS.

**Podporované platformy:**

- Android
- iOS (ale zařízení musí být MFi certifikované)
- Prohlížeč

**Omezení:**

- Android SDK musí být levl 22 nebo menší
- Nový systém oprávnění pro SDK23 (Android 6.0) ještě není implementován
- Není možnost připojit se k více zařízením, nicméně od verze 0.9.5 podporuje více rozhraní na jednom zařízení [17]

## 4.5 Cordova-plugin-ble-central

Jde o jeden z nejpopulárnějších pluginů pro Bluetooth Low Energy. Výše jmenovaný plugin nám dovoluje komunikovat mezi telefonem a Bluetooth Low Energy periferiemi.

**Podporované platformy:**

- Android 4.3 nebo vyšší
- iOS

Připojení k více periferiím je podporováno. K zařízení se přistupuje pomocí UUID [18].

## 4.6 Cordova-plugin-bluetoothle

Jedná se o plugin, který má built-in TypeScript deklarace. Plugin s built-in TypeScript deklaracemi slouží ke komunikaci mezi Bluetooth Low Energy zařízeními a zařízeními, která mají Android, iOS nebo Windows.

**Požadavky:**

- Systém musí mít Cordovu ve verzi 5.0.0 nebo vyšší
- Minimální verze Android je 4.3 a minimální Android API Level 23
- U iOS se jedná o iOS 10 a vyšší
- Podporovaná verze Windows Phone je 8.1
- Hardware zařízení musí být certifikováno pro Bluetooth LE. i. e. Nexus 7

**Omezení:**

- Při odpojení a rychlém opětovném připojení se mohou vyskytnout problémy na Androidu. Je potřeba implementovat malé zpoždění.
- OS X je stále na experimentální úrovni [19]

## 4.7 Cordova-plugin-ble

Poslední populární plugin implementuje podporu Bluetooth Low Energy pro Android, iOS a Windows 8.1 (částečná podpora). Jedná se o starší plugin, proto některé operace nemusí být funkční. Plugin umožňuje scanovat Bluetooth Low Energy zařízení na pozadí, navazovat spojení, čtení a zápis hodnot, příjem upozornění o změnách hodnot, měření síly signálu RSSI (signal strength) a experimentální podpora pro periferní mód na Androidu. Po nejnovějším updatu je možné dávat do funkcí objekty [20].

# Kapitola 5

# Návrh testů

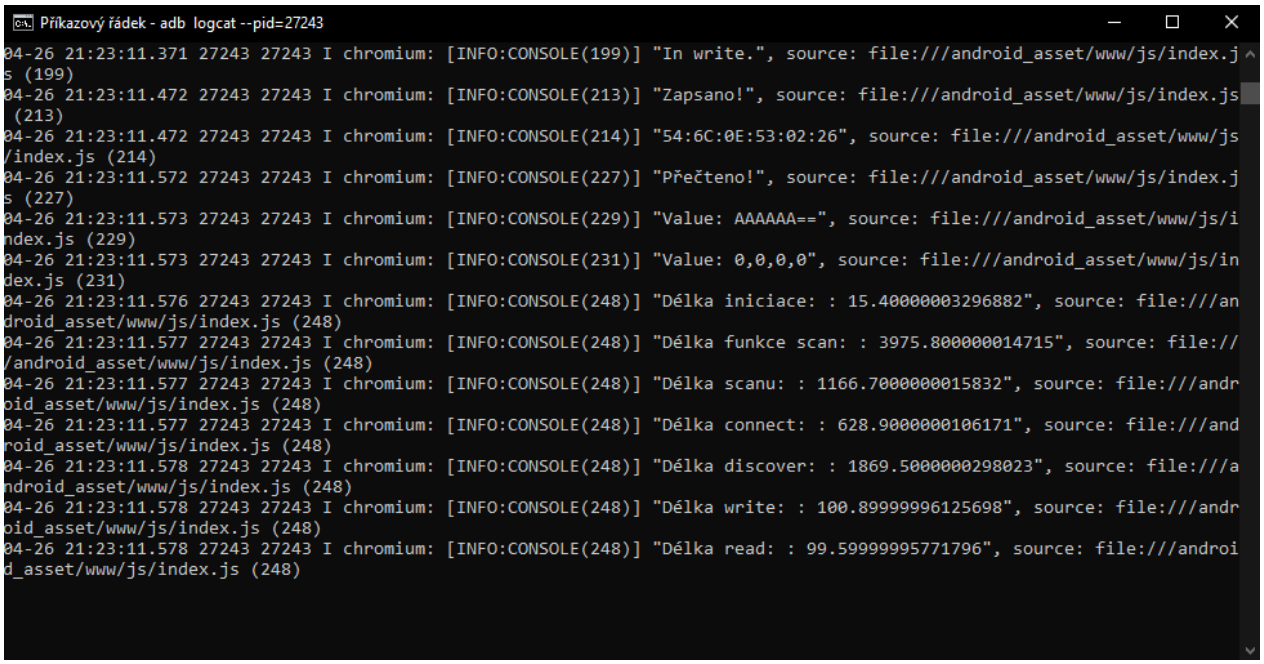
Pro realizaci testů jsem si vybral Performance API pro Javascript. Jmenované API definuje rozhraní, které podporuje měření různých parametrů s vysokou přesností. Rozhraní podporuje velké množství případů užití, jako jsou výpočet snímkových frekvencí nebo srovnávací testy. Srovnávacími testy je myšleno například srovnávání časů na načtení prostředků.

Pokud chceme využít jmenované API stačí v kódu napsat slovo performance a následně zvolit potřebnou metodu. V mé práci jsem se zaměřil na metody: *performance.mark()* a *performance.measure()*. *performance.mark()* nám zaznamená časové razítko s určitým jménem, které následně využijeme jako identifikátor pro *performance.measure()*. Metoda measure nám porovná dvě časová razítka, která specifikujeme podle námi zvolených identifikátorů. Zde musíme opět výsledné hodnotě dát náš identifikátor. Následně si rozdíl těchto razítek uloží do paměti a až potřebujeme, můžeme si je zavolat pomocí *performance.getEntriesByType()* do parametru metody si dáme measure, jelikož chceme získat všechny hodnoty z rozdílů. Výsledné časy jsou získané v milisekundách, což nám zaručuje velkou přesnost. Performance API nám umožňuje také měřit množství paměti využité aplikací prostřednictvím *performance.memory*, nicméně v případě hybridních aplikací by měřilo pouze paměť alokovanou v JavaScriptu a výsledky by tak nebyly porovnatelné. Ovšem toto nestandardní rozšíření Performance API se nedoporučuje používat [21].

Existuje několik možností, jak výsledky získávat ze zařízení. Jednou z nich je výpis do konzole, kterého lze dosáhnou pomocí konzolové metody *log()*. Zapisujeme ji *console.log()* a do parametru uvádíme proměnnou, kterou chceme vypsat. Popsané řešení je ideální pro programátory, kteří jsou schopni se k těmto informacím lehce dostat. Pro běžného uživatele to není příliš použitelné. Příklad výpisu je v ukázce na Obrázku 5.1.

Proto můžeme využít jiný postup, kterým je vypsání hodnot na display. Získaný výstup na display je asi nejlepší, protože se k informacím dostane kdokoliv a současně již není potřebné cokoliv dalšího provádět.

Poslední možností je export hodnot do samostatného souboru, kterou obyčejný uživatel běžně nezvolí. Hodnoty nejsou dostupné hned, což je pro tento typ uživatele nezajímavé a odrazující. Uve-




Obrázek 5.1: Výpis hodnot do konzole u pluginu cordova-plugin-bluetoothle

dený typ uživatele může mít na svém zařízení k uložení dat omezené množství paměti, popřípadě nechce hledat v adresáři soubor s hodnotami. Vhodnější variantou je z výše uvedených důvodů druhá možnost řešení.

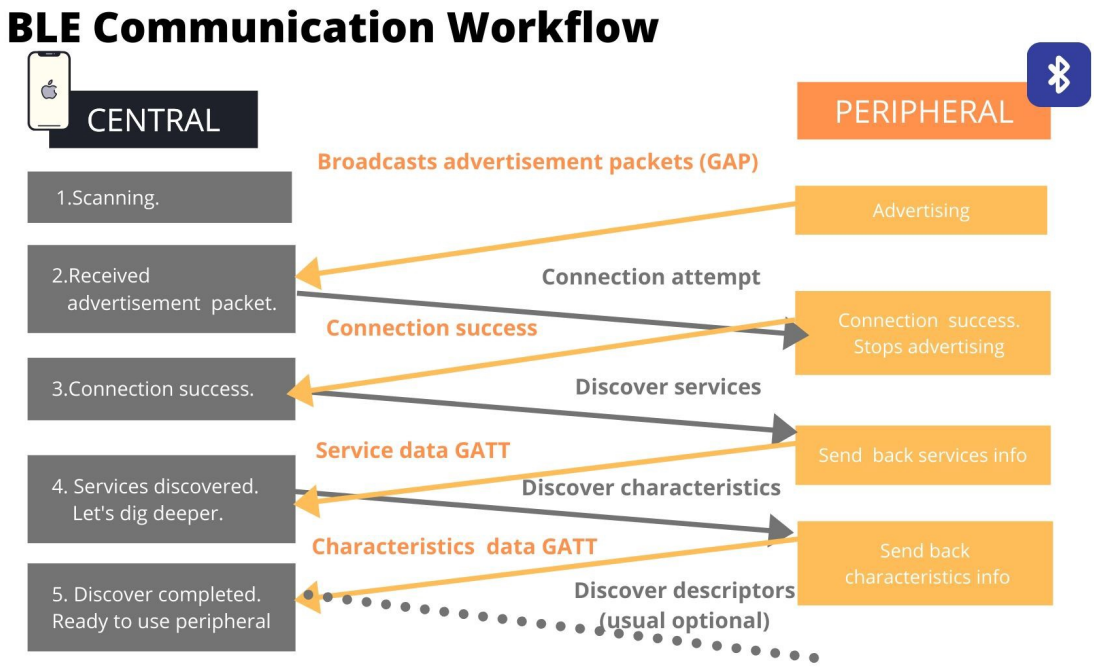
Jelikož budeme testovat rychlost Bluetooth API, tak jsem se rozhodl svou práci rozdělit podle metod, které jsou pro práci s Bluetooth stěžejní a které můžeme najít u všech API. Některá API ovšem mají specifické funkce. U nich čas změříme, ale už jej následně nebudeme porovnávat.

U každé testované funkce jsem se rozhodl, že test provedu desetkrát, abych omezil externí vlivy. Následné výsledky testů zprůměruji a výslednou hodnotu budu uvádět jako konečnou.

Rozhodl jsem se, že budu pracovat s těmito API: Web Bluetooth API, cordova-plugin-ble-central a cordova-plugin-bluetoothle.

## 5.1 Testy

V testu se nejprve zaměříme na dobu, po kterou bude trvat, než Bluetooth API zjistí dostupnost a přítomnost Bluetooth adaptéru. Metoda rovněž detekuje, zda máme Bluetooth zařízení aktivováno. Výše zmíněná metoda se nachází u všech tří Bluetooth API. Při spuštění testu se měří doba, než dané metody projdou zařízení a zjistí, zda zařízení obsahuje Bluetooth adaptér. Pokud zařízení Bluetooth adaptér obsahuje, zjišťuje se, zda je aktuálně zapnutý či vypnutý. Výslednou informaci nám metoda vrátí a test se ukončí.




Obrázek 5.2: Diagram Bluetooth komunikace, převzato z [22]

Dalším testem budeme zjišťovat, jak dlouho trvá vyhledat periferní zařízení. Jde o další klíčovou funkci k ovládání Bluetooth. Na Obrázku 5.2 můžeme vidět, že první blok u centrálního zařízení je skenování. Zařízení se snaží zachytit, zda nějaká Bluetooth periferie nevysílá informaci, že je možné se k ní připojit. U všech pluginů můžeme specifikovat, jaké služby má hledané zařízení obsahovat. Najdeme ji opět u všech tří Bluetooth API.

V dalším kroku budeme testovat dobu navázání spojení s periferním zařízením. Do naskenovaného zařízení, ke kterému jsme se chtěli připojit, můžeme odeslat požadavek na pokus o připojení. Na Obrázku 4.1 je znázorněn jako Connection attempt. Následně čekáme, zda nám periferie vrátí zprávu o úspěšném či neúspěšném připojení. Po zdárném připojení je možné přistupovat ke službám a charakteristikám.

Níže uvedený test zjišťuje, jak dlouho trvá přistoupit k primární, námi zvolené, službě. Jiné Bluetooth API vyžadují nejprve prohledání zařízení a nalezení služeb, k nimž můžeme přistoupit. Po tzv. objevení služby můžeme přistoupit k její charakteristice. V testu se podíváme, jak dlouho trvá, než se projdou a zpřístupní všechny služby. U těchto dvou testů budeme ověřovat rozdílné metody, které nám umožňují přístup ke službám a charakteristikám. Rozhodl jsem se je zařadit do společného testu. Na Obrázku 5.2 postup ilustruje šipka s označením *Discover services*. Následně se nám vrátí informace o zpřístupněných službách.

V dalším testu zjistíme, jak dlouho trvá, než se dostaneme do určité, námi zvolené charakteristiky. Zde uvádíme, jaké službě daná charakteristika náleží. K charakteristice se připojujeme,

abychom mohli do zařízení zapisovat nebo z něj číst hodnoty. Provedu dvakrát výše uvedený test. Důvodem je, že k jedné charakteristice přistupuji pro zápis a druhé pro čtení.

Následně přistoupíme k další důležité funkci, kterou je zápis do charakteristiky. Otestujeme ji a bude nás zajímat srovnání, jak dlouho trvá zápis hodnoty. Pokud chceme zapsat hodnotu do charakteristiky, musíme vždy uvést, do jaké charakteristiky chceme zapisovat. Je potřebné uvést hodnotu ve správném datovém typu. Po odeslání naší hodnoty do zařízení, se nám vrátí Promise, o kterém si informace sdělíme v další kapitole. Promise nás informuje, zda se zápis hodnoty do periferie zdařil nebo došlo k chybě.

V posledním testu budeme měřit čas, než přečteme hodnotu z charakteristiky. Do námi zvolené charakteristiky pošleme Promise s dotazem o zaslání očekávané hodnoty. Pokud zařízení disponuje daty, která požadujeme, vrátí nám je opět v podobě Promise. Musíme kontrolovat, jakého datového formátu jsou příjímaná data a následně, pokud bude nutné, je převedeme do námi zvoleného datového typu, který nám umožní data interpretovat.

# Kapitola 6

# Implementace navržených testů

V této kapitole rozebereme jak implementaci samotného kódu pro komunikaci s Bluetooth periferií, tak i navržené testy, které jsme si popsali výše. Projdeme si tedy postupně kód pro Web Bluetooth API a také pro oba Apache Cordova pluginy.

## 6.1 Implementace testů pro Web Bluetooth API

Dokumentace Web Bluetooth API je dobře zpracovaná. Zároveň je velice oblíbené u amatérských vývojářů. Web Bluetooth API je podporován v nejužívanějším prohlížeči Chrome, což jen napomáhá jeho popularitě. Lze také dohledat dostatečné množství tutoriálů, které práci s API popisují. U Web Bluetooth API je komfortní psaní kódu v asynnchroní formě. Při využití možnosti, kdy nečekáme na Promise, se kód začal stávat méně čitelným.

### 6.1.1 Implementace testu dostupnosti Bluetooth

Jak můžeme vidět v kódu níže, využíváme metodu *getAvailability()*. Tato metoda nám vrací Boolean hodnotu, která nabývá hodnoty true pokud zařízení má Bluetooth dostupné a zapnuté. Pokud zařízení nemá Bluetooth zapnuté, nebo jím nedisponuje, vrátí nám hodnotu false. Podle toho se nám vypíše oznámení o dostupnosti, popřípadě nedostupnosti Bluetooth. Zároveň zde můžeme vidět první užití Performance API. Konkrétně se jedná o metodu *mark()*. Do parametru této metody vložíme identifikační řetězec. První užití metody mark() s parametrem startAvailability nám udává začátek testu a druhý užitý mark() s parametrem endAvailability konec testu. Ukázkové vyhodnocení můžeme vidět v bodě 6.1.8.

```
performance.mark('startAvailability');

navigator.bluetooth.getAvailability()
.then(isBluetoothAvailable => {
  document.getElementById("bluetooth_status").innerHTML =
  (`Bluetooth is ${isBluetoothAvailable ? 'available' : 'unavailable'}`);
});

performance.mark('endAvailability');
```

### 6.1.2 Implementace testu vypsání dostupných zařízení

Následně v aplikaci využijeme metodu *requestDevice()*. Tato metoda se dotazuje všech dostupných Bluetooth zařízení, zda je možné se k nim připojit. Pokud ano, vypíšou se nám zařízení do seznamu, z kterého si můžeme vybrat naše hledané zařízení. Parametry této metody jsou volitelné. Prvním možným parametrem je *filters[]*. Do pole můžeme dát jednu nebo více z těchto tří hodnot: UUID námi hledané služby, jméno zařízení a prefix jména zařízení. Dalším parametrem metody requestDevice() je *optionalServices[]*. Zde máme pole, do kterého doplníme UUID služeb, které budeme chtít využívat (převzali jsme je z Obrázku 3.2). Posledním parametrem je Boolean indikátor *acceptAllDevices*. Pokud použijeme tento parametr a nastavíme jeho hodnotu na true, budeme mít možnost zobrazovat všechna zařízení. V mém kódu jsem užil kombinace parametrů acceptAllDevices a optionalServices. V optionalServices jsem uvedl UUID služby a všech jejích charakteristik, s kterými jsem v průběhu pracoval. Pokud jsem neuvedl službu, ke které jsem chtěl přistoupit, nebylo mi dovoleno do ní vstoupit.

```
  dev = await navigator.bluetooth.requestDevice({
    acceptAllDevices: true,
    optionalServices: [0x2800,'f000aa00-0451-4000-b000-000000000000',
    'f000aa01-0451-4000-b000-000000000000','f000aa02-0451-4000-b000-000000000000']
  })
```

### 6.1.3 Implementace testu navázání spojení k zařízení

V této části se připojíme k zařízení, které jsme si vypsali pomocí requestDevice(). Využíváme konstrukce *dev.gatt.connect()*. Periferní zařízení se stane GATT serverem. Využívá se zde rozhraní BluetoothRemoteGATTServer. Není třeba zde udávat žádný parametr. Metoda nám vrátí Promise. Promise je objekt, který má hodnotu, která může nabývat jeden z těchto stavů: nevyřízeno (což je

počáteční stav), splněno (objeví se, pokud se operace úspěšně dokončí), odmítnuto (znamená, že operace selhala) [23].

```
gatt = await dev.gatt.connect();
```

### 6.1.4 Implementace testu připojení se k službě

I zde vycházíme z rozhraní BluetoothRemoteGATTServer. Metoda pro přístup do služby se nazývá *getPrimaryService()* a jejím parametrem je UUID služby, ke které se chceme připojit. Metoda nám vrátí Promise, který obsahuje přímo volanou službu. UUID je potřebné vložit jako řetězec a všechna písmena musí být malá.

```
service = await gatt.getPrimaryService("f000aa00-0451-4000-b000-000000000000");
```

### 6.1.5 Implementace testu připojení se k charakteristice

Máme zde velice podobnou konstrukci kódu jako v předchozí podkapitole. Ale i tak zde najdeme několik rozdílů. Hlavní odlišností je jméno metody *getCharacteristic()*. Tentokrát se jedná o metodu rozhraní BluetoothGATTService. Můžeme vidět, že do parametru metody zase vkládáme UUID, ale tentokrát použijeme UUID charakteristiky. Nicméně opět je nám vrácen Promise, tentokrát od námi volané charakteristiky. Podobně jako u metody *getPrimaryService* musí být řetězec psán malým písmem. V práci jsem volal dvě charakteristiky. Do první jsem následně zapisoval hodnotu a z druhé vyčítal hodnoty. Rozhodl jsem se změřit obě charakteristiky.

```
tempStart = "f000aa02-0451-4000-b000-000000000000";
ch2 = await service.getCharacteristic(tempStart);
tempStart1 = "f000aa01-0451-4000-b000-000000000000";
characteristics = await service.getCharacteristic(tempStart1);
```

### 6.1.6 Implementace testu zápisu dat do zařízení

Pro zápis do charakteristiky využíváme metodu *writeValue()*. Do parametru vkládáme hodnotu, která je reprezentovaná binárními daty. Ve své práci využívám Uint8Array, které reprezentuje pole bajtů. Nastavíme bajt na hodnotu 1. Následně hodnotu zaznamenám do charakteristiky. Po zapsání je vrácen Promise, který oznámí, zda se zápis podařil či nikoliv.

```
ar = new Uint8Array(1);
ar[0] = 1;

await ch2.writeValue(ar);
```

### 6.1.7 Implementace testu čtení dat ze zařízení

Nyní když víme, jak zařízení vyhledat, připojit se k němu a zapnout senzor, potřebujeme přečíst hodnotu ze zařízení. K tomu slouží metoda *readValue()*. Jedná se o metodu z rozhraní BluetoothRemoteGATTCharacteristic. Parametr metody se zde neudává. Metoda nám vrací Promise, který má hodnotu vyčtenou ze zařízení. Vrácené hodnoty jsou reprezentovány binárními daty. Proto musíme následné hodnoty převést.

```
val = await characteristics.readValue();
```

### 6.1.8 Implementace zpracování měření

Zde jsem uvedl příklad, jak následné mark() metody zpracováváme pomocí metody measure().Prvním parametrem metody *measure()* je náš vlastní identifikátor. Druhým parametrem je identifikátor metody mark(), která měřený úsek začíná. Posledním parametrem je identifikátor metody mark(), která měřený úsek ukončuje. Metoda measure hodnoty uloží pod námi zadaný identifikátor. Abychom se k hodnotám dostali, využijeme metodu *getEntriesByType()*, kde do parametru uvedeme, že chceme právě metodu measure. Následně můžeme každý prvek vypsat, jak nám ukazuje příklad. Na závěr vymažeme všechny hodnoty mark a measure z paměti, abychom je mohli v budoucnu dále používat a dostávali hodnoty, o které očekáváme.

```
performance.measure("Delka zjisteni dostupnosti bluetooth: ",
"startAvailability", "endAvailability");

const measurment = performance.getEntriesByType("measure");

measurment.forEach(measureItem => {
console.log(`${measureItem.name}: ${measureItem.duration}`);

performance.clearMarks();
performance.clearMeasures();
```

## 6.2 Implementace cordova-plugin-central

S pluginem cordova-plugin-central se mi pracovalo nejlépe. Dokumentace je přehledná a dobře popsaná. Obsahuje i rychlé příklady, jak danou metodu užívat. Parametery metod jsou přehledně a podrobně popsány. Až na problém při zápisu, kdy je za hodnotu potřebné dosadit *.buffer*, který v dokumentaci nebyl dostatečně označen, bylo užití pluginu bezproblémové.

### 6.2.1 Implementace testu dostupnosti Bluetooth

Pro testování dostupnosti Bluetooth využíváme funkci *isEnabled()*. Pokud je Bluetooth dostupný, provede se úspěšná callback funkce. V našem případě se vypíše, že Bluetooth je dostupný. Pokud by Bluetooth byl nedostupný, provede se neúspěšná funkce, která vypíše, že Bluetooth není dostupný. Zároveň u daného pluginu máme možnost zjistit, jestli jsou povoleny služby určování polohy. Zde využíváme metodu *isLocationEnabled()*, která vrací buď úspěšné nebo neúspěšné volání a podle výsledku nám vypíše text, zda je určování polohy povolené nebo zakázané.

```
ble.isEnabled(
    function() {
        document.getElementById("bluetooth_status").innerHTML =
        ("Bluetooth is available");
    },
    function() {
        document.getElementById("bluetooth_status").innerHTML =
        ("Bluetooth is unavailable");
    }
);

ble.isLocationEnabled(
    function() {
        document.getElementById("location").innerHTML =
        ("Location is available");
    },
    function() {
        document.getElementById("location").innerHTML =
        ("Location is available");
    }
);
```

### 6.2.2 Implementace testu vypsání dostupných zařízení

V testu se zaměříme na naskenování a objevení Bluetooth periferií. Používáme k tomu metodu *scan()*, kde do parametrů můžeme vložit seznam služeb, které očekáváme u zařízení. Pokud bychom chtěli vypsat všechna zařízení, pak použijeme *[]* - prázdné pole. Dalším parametrem je počet sekund, jak dlouho má skenování běžet. V mém případě se jedná o 5 sekund. Následně nám metoda vrátí úspěšné volání, při kterém budeme pokračovat v kódu. Při neúspěšném volání vypíšeme chybu.

```
ble.scan(['AA80'], 5, function(device) {
```

```
}, e=>console.error(e));
```

### 6.2.3 Implementace testu navázání spojení k zařízení

Pro navázání spojení se využívá metoda *connect()*. U metody connect je potřebné uvést do parametrů UUID nebo MAC adresu zařízení. V mé práci jsem se rozhodl pro MAC adresu. Bylo to z důvodu znalosti adresy z předchozího vypsání zařízení a jednoduchému přístupu k ní. Následně nám metoda vrací dvě callback funkce. Jednu pro úspěšné připojení se k zařízení. Daná funkce pokračuje dále v získávání hodnot. Druhá funkce při neúspěšném připojení k zařízení vypíše chybu.

```
ble.connect(device.id, function(device) {
}, e=>console.error(e));
```

### 6.2.4 Implementace testu zápisu dat do zařízení

K zápisu do zařízení využijeme funkce *write()*. Zde zadáváme více parametrů než v předchozích metodách. Prvním je UUID nebo MAC adresa zařízení, druhým parametrem je UUID služby a třetím parametrem je UUID charakteristiky, kam budeme zapisovat. Posledním parametrem jsou data ve formátu binárních dat. Níže je ukázka kódu s ukázkou testovacích mark i připravení hodnoty k zápisu. Následuje zápis, kde udávám MAC adresu a UUID služby a její příslušné UUID charakteristiky. Jak můžeme vidět, musíme zde přistoupit k atributu .buffer obsahující data, která chceme zapsat. V dokumentaci je tato skutečnost skryta a je lehké ji přehlédnout. Následně nám metoda opět vrátí dvě callback funkce. Při úspěšném zápisu pokračuji ve čtení hodnoty. Při neúspěšném zápisu se vypíše chybu.

```
ar = new Uint8Array(1);
ar[0] = 1;

ble.write(device.id, 'f000aa00-0451-4000-b000-000000000000',
"f000aa02-0451-4000-b000-000000000000", ar.buffer, function(device2){
}, e=>console.error(e));
```

### 6.2.5 Implementace testu čtení dat ze zařízení

O parametr méně máme u funkce *read()*, kterou využíváme pro čtení hodnot ze zařízení. Zůstal nám zde parametr, kde udáváme UUID nebo MAC adresu zařízení. Stejně tak zde máme parametry, kde každá služba a charakteristika má vlastní UUID, ze které chceme hodnotu číst. Následně s hodnotou zacházíme stejně jako u Web Bluetooth API.

```
ble.read(device.id, 'f000aa00-0451-4000-b000-000000000000',
"f000aa01-0451-4000-b000-000000000000", function(val){
}, e=>console.error(e));
```

## 6.3 Implementace cordova-plugin-bluetoothle

U pluginu cordova-plugin-bluetoothle je potřebné zohlednit životní cyklus aplikace, kterou budeme tvořit. Musíme začít inicializací lokálního rozhraní a až následně můžeme vypisovat zařízení a připojovat se k nim. Pokud chceme přistupovat ke službám, musíme je všechny nejdříve objevit a až poté s nimi můžeme pracovat. Zároveň nikde není zdůrazněno, že je celá dokumentace rozdělena zvlášť pro centrální Bluetooth zařízení a zvlášť periferní Bluetooth zařízení. Díky těmto nedokonalostem se mi ze začátku nedařilo plugin korektně využít. I přesto bych chtěl ocenit připravené příklady, kde bylo vidět, že si je autor vědom vyšší složitosti svého pluginu, a proto byly manuály zpracované podrobně.

### 6.3.1 Implementace testu dostupnosti Bluetooth

U daného testu zkoumáme průběh metody *initialize()*. Tato metoda inicializuje Bluetooth adaptér a pokud vše proběhne úspěšně, značí to, že je Bluetooth dostupný. Vše je voláno v konstrukci Promise. Do parametrů udáváme request na hodnotu true, která nám značí, že uživatel bude v případě vypnutého Bluetooth adaptéru vyzván k jeho zapnutí. Druhým parametrem je statusReceiver, který nastavíme na false. Tento parametr udává, zda má uživatel obdržet upozornění o stavu Bluetooth.

```
new Promise(function (resolve) {

    bluetoothle.initialize(resolve, { request: true, statusReceiver: false });

}).then(initializeSuccess, handleError);
```

### 6.3.2 Implementace testu vypsání dostupných zařízení

Následně po inicializaci můžeme konečně provést hledání Bluetooth periferií. Používáme k tomu metodu *startScan()*. Zde do parametru uvádíme služby, které od zařízení očekáváme. Pole můžeme nechat prázdné. V tom případě by nám byla vyhledána všechna zařízení. Metoda nám vrací callback. V případě jeho úspěchu pokračujeme dále. Při neúspěšném callbacku vypíšeme chybu.

```
bluetoothle.startScan(startScanSuccess, handleError, { services: ['AA80'] });
```

### 6.3.3 Implementace testu navázání spojení k zařízení

Pro navázání spojení používáme metodu *connect()*. Do parametru metody se udává MAC adresa zařízení, kterou jsme získali v průběhu metody startScan(). Callback funkce jsou podobné jako u předchozí metody. Při úspěšné callback funkci budeme pokračovat dále. Při neúspěšné callback funkci vypíšeme chybu.

```
new Promise(function (resolve, reject) {

    bluetoothle.connect(resolve, reject, { address: address });

}).then(connectSuccess, handleError);
```

### 6.3.4 Implementace testu zjišťování dostupných služeb

Abychom mohli dále pokračovat a přistupovat ke službám, musíme je nejdříve zjistit dostupné služby. To se provádí metodou *discover()*. Daná metoda projde celé zařízení a všechny služby nám zpřístupní. Do parametru metody musíme opakovaně vložit MAC adresu zařízení. Opět nám callback funkce, které se vrátí, mohou skončit úspěšně nebo neúspěšně. V případě úspěšného zjištění budeme pokračovat, pokud nastane problém vypíšeme chybu.

```
    new Promise(function (resolve, reject) {

        console.log("Discovering...");

        bluetoothle.discover(resolve, reject,
            { address: address });

    }).then(discoverSuccess, handleError);
```

### 6.3.5 Implementace testu zápisu dat do zařízení

Pokud se nám povedlo úspěšně objevit všechny potřebné služby, tak můžeme spustit senzor zapsáním hodnoty do zařízení. K uvedenému kroku je potřebné využít funkce *write()*. Do parametru se kromě naší obvyklé MAC adresy zařízení udává i hodnota, která musí být v kódování Base64. Pro převod máme v pluginu připravenou metodu bytesToEncodedString(). Dalšími parametry jsou UUID služby a UUID charakteristiky. Narozdíl od Web Bluetooth API a cordova-plugin-central zde můžeme použít velká písmena.

```
ar = new Uint8Array(1);
ar[0] = 1;
ar1 = bluetoothle.bytesToEncodedString([0x01]);
console.log("Base64 - " + ar1);

console.log("In write.");

writeParamsObj = {"address" : result.address, "value" : ar1,
                      "service" :'F000AA00-0451-4000-B000-000000000000',
                      "characteristic" :"F000AA02-0451-4000-B000-000000000000"
                          };

bluetoothle.write(writeSuccess, handleError, writeParamsObj);
```

### 6.3.6 Implementace testu čtení dat ze zařízení

Pokud chceme přečíst hodnotu ze zařízení, využijeme metodu *read()*. Máme zde podobné parametry jako v metodě write. Prvním parametrem je MAC adresa zařízení. Druhým parametrem je UUID služby a třetím parametrem je UUID charakteristiky, která nám hodnotu poskytne. Výstup je zakódován v Base64. Pro převedení hodnoty na 8 bitové pole použijeme metodu encodedStringToBytes().

```
readParamsObj = {"address" : result.address,
                      "service" :'F000AA00-0451-4000-B000-000000000000',
                      "characteristic" :"F000AA01-0451-4000-B000-000000000000"
                          };

bluetoothle.read(readSuccess, handleError, readParamsObj);
```

# Kapitola 7

# Testovací zařízení a vyhodnocení testů

Testy proběhly na desktopu a na trojici mobilních zařízení. Testy Web Bluetooth API probíhaly v Google Chrome. Původně jsem chtěl otestovat Web Bluetooth API i ve výchozím prohlížeči na mobilních zařízeních. To bohužel nebylo možné, protože výchozí prohlížeč nepodporuje Web Bluetooth API viz Obrázek 4.1. Testy pluginů Cordovy probíhaly v systémovém WebView, které používá stejnou komponentu jako výchozí prohlížeč systému.

Nejprve si sdělíme informace o zařízeních a v druhé části této kapitoly si ukážeme výsledky naměřených testů.

## 7.1 Testovací prostředí

Zařízení, kterým se budeme zabývat jako první bude desktop, který běží na operačním systému Windows 10. Dále si seřadíme zařízení Android chronologicky podle jejich stáří od nejstarší po nejnovější.

### 7.1.1 Dell G3 15 Gaming

Posledním zařízením je Dell G3 15 Gaming. Je to notebook se čtyřjádrovým procesorem Intel Core i5 8. generace - 8300HQ, jehož jádra jsou taktovaná na hodnotu 2.3GHz. Operační paměť má velikost 8 GB DDR4. Nalezneme zde 2 uložiště. Prvním je 128 GB SSD disk a druhým 1 TB pevný disk. Celý systém běží na operačním systému Windows 10. Zařízení disponuje Bluetooth standardem 5.0 [24].

Testování probíhalo v prohlížeči Google Chrome, kde jsme testovali Web Bluetooth API. Na Dell notebooku se během testu vyskytl jediný problém. Při úvodním připojení se k perifernímu zařízení nám byla vypsána chyba, že se k zařízení nelze připojit. Následně se nám podaří připojit, a i po odpojení jsme schopni zařízení připojit bez problémů. Zbytek testu proběhne bez problémů.

### 7.1.2 Huawei P8 Lite

Druhým testovacím zařízením je Huawei P8 Lite. Jde o starší mobilní telefon. Procesor je HiSillicon Kirin 620 a je vyrobený 28nm procesem. Procesor má osm jader Cortex-A53, která jsou taktována na 1,2 GHz. Operační paměť má velikost 2 GB a interní paměť má kapacitu 16 GB. Jelikož se jedná o starší model, disponuje pouze Bluetooth 4.0. Během testu byla verze systému Android 6.0 a verze nadstavby EMUI byla 4.0.3 [25].

Během testování se objevil problém s funkčností Cordova pluginů, které nejspíš nebyly kompatibilní se zařízením. Uvedená skutečnost byla důvodem, že se měření podařilo uskutečnit pouze u Web Bluetooth API. Myslím si, že výsledky měření hodnot ze zařízení nemají relevantní vypovídající schopnost. Použité mobilní zařízení je opravdu staré a jeho funkčnost je ovlivněna vysokou mírou opotřebení a zastaráním. Do testů jsem jej zařadil spíše jen ze zvědavosti. Mnohem důležitější jsou výsledky měření z dalších zařízení.

### 7.1.3 Xiaomi Redmi 4X

Nejedná se o nejnovější, ale ani o nejstarší testovací zařízení. Procesor je Qualcomm Snapdragon 435 s osmi jádry Cortex-A53 a je taktován na 1,4 GHz. Operační paměť má velikost 3 GB a vnitřní uložiště dosahuje velikosti 32 GB. Verze systému Android v době testu byla 7.1.2 N2G47H a byla použitá nadstavba MIUI Global 11.0.2 [26].

Na Xiaomi zařízení se mi všechny testy podařilo spustit bez problémů. Před spuštěním bylo potřebné povolit instalaci pomocí USB, která byla defaultně povolená u předchozích dvou zařízení

### 7.1.4 Honor 10

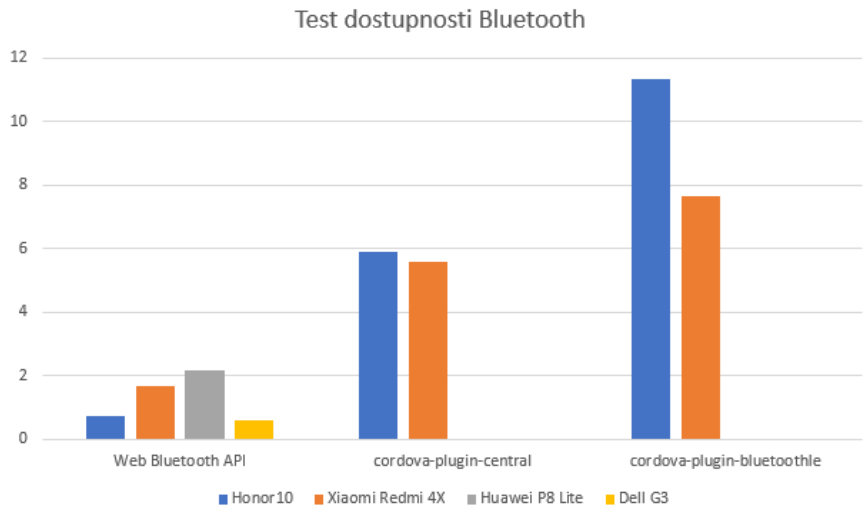
Prvním testovacím zařízením se stal můj osobní smartphone Honor 10. Procesorem je HiSilicon Kirin 970, který byl uveden na trh v roce 2017. Má procesor s osmi jádry vyrobenými 10nm procesem. Je založen na architektuře ARM. První polovina jader je typu Cortex-A73 a je taktována na 2.36 GHz. Druhá polovina jader Cortex-A53 je taktována na frekvenci 1.8 GHz. Smartphone disponuje 4 GB operační paměti a interním uložištěm s velikostí 64 GB. Je zde použitý Bluetooth 4.2. Během testu byla verze systému Androidu 10 a verze nadstavby EMUI byla 10.0.0 [27].

U tohoto zařízení nebyl během testu žádný problém. Testy proběhly bezchybně.

## 7.2 Vyhodnocení testů dostupnosti Bluetooth

U prvního testu, který zjišťuje dostupnost Bluetooth, jsem naměřil většinu hodnot v jednotkách milisekund, jak můžeme vidět na Obrázku 7.1. Pouze u pluginu cordova-plugin-bluetoothle na zařízení Honor 10 jsem naměřil 11 ms, což je zvláštní, protože se jednalo o výkonnější zařízení. U Web Bluetooth API jsem obdržel očekávaný výsledek, kde výkonnější zařízení bylo nejrychlejší a méně

výkonné zařízení nejpomalejší. U cordova-pluginu-central nezáleželo na výkonosti zařízení, výsledky jsou si ale i tak dosti podobné.



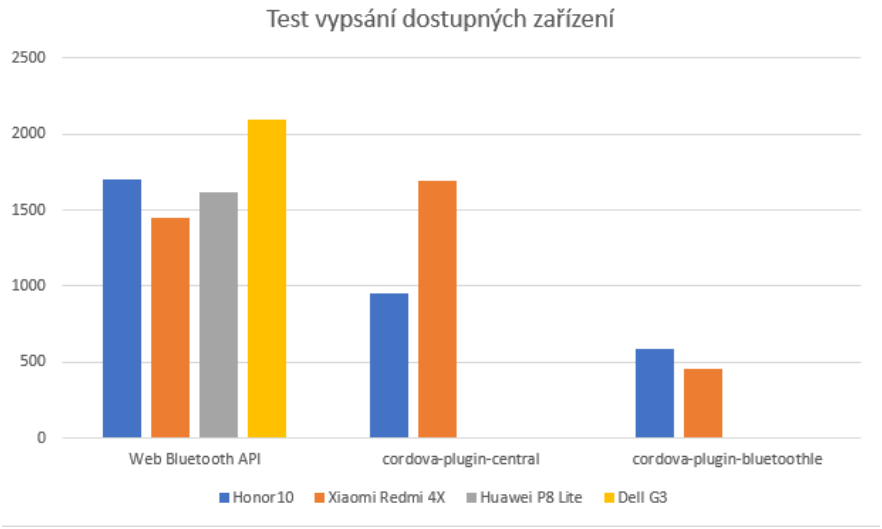

Obrázek 7.1: Test dostupnosti Bluetooth

## 7.3 Výsledek testů vypsání dostupných zařízení

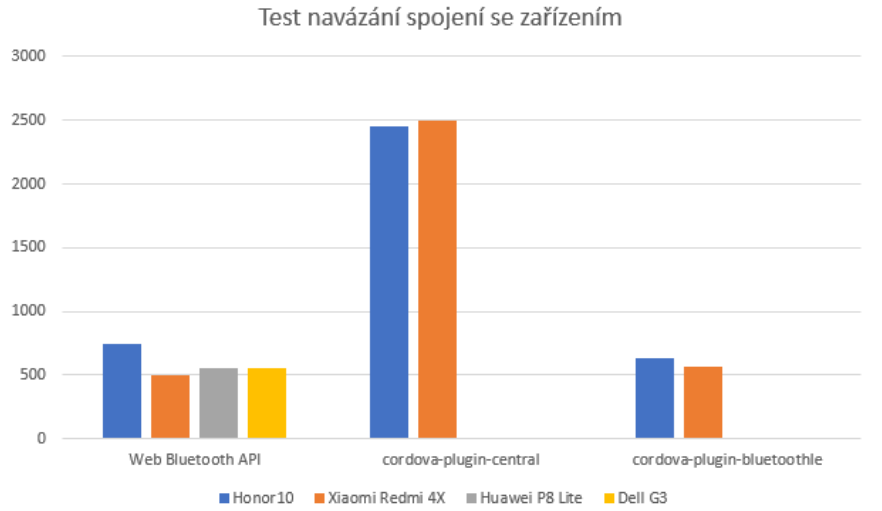
U druhého testu jsem měřil výpis dostupných zařízení. Grafický výstup můžeme vidět na Obrázku 7.2. Jednalo se o náročnější operaci, proto nám výsledky vycházely až v jednotkách sekund. Poměrně konzistentní výkon bez závislosti na zařízení můžeme vidět u Web Bluetooth API. Snad jen zařízení Dell má trochu delší čas vykonání metody. Současně si můžeme povšimnout, že doba trvání měření roste s výkonnějším zařízením. U pluginu cordova-plugin-central bylo dvojnásobně rychlejší výkonnější zařízení. U posledního pluginu můžeme vidět, že bylo lehce rychlejší méně výkonné zařízení. Je pravděpodobné, že testy byly ovlivněni interními parametry skenování Bluetooth v rámci jeho implementace

## 7.4 Výsledek testů navázání spojení k zařízení

Následující test zkoumal, jak dlouho trvá navázat spojení s námi vybraným zařízením. U Web Bluetooth API jsou výsledky ve stovkách milisekund, viz Obrázek 7.3. Honor 10 je zde nejpomalejší. Celkové výsledky u daného API si byly podobné. U cordova-plugin-bluetoothle se jednalo o jednotky sekund. výkonnější zařízení bylo nepatrně rychlejší. V posledním pluginu cordova-plugin-bluetoothle se jednalo o stovky milisekund. Méně výkonné zařízení zde bylo rychlejší přibližně o sto milisekund.



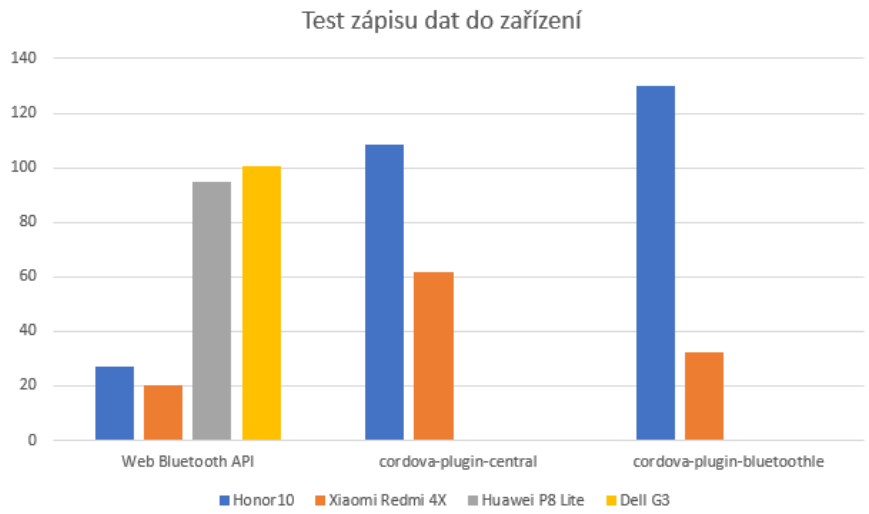

Obrázek 7.2: Test vypsání dostupných zařízení




Obrázek 7.3: Test navázání spojení k zařízení

## 7.5 Výsledek testů zápisu dat do zařízení

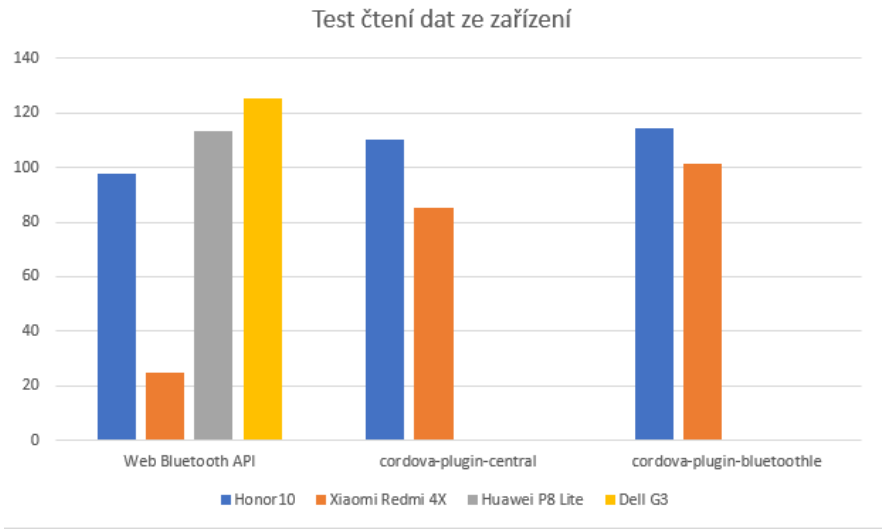
Dalším testem byl zápis dat do zařízení, které nalezneme na Obrázku 7.4. Zde jsme se pohybovali do 200 milisekund. Některá zařízení zvládla provést zápis i během desítek milisekund. U Web Bluetooth API jsem získal podobné výsledky jako u zařízení Honor 10 a Xiaomi Redmi 4X. U zařízení Huawei P8 Lite a Dell G3 15 Gaming byl průběh metody přibližně 4krát delší. U pluginu cordova-plugin-central bylo méně výkonné zařízení dvakrát rychlejší a u pluginu cordova-plugin-bluetoothle dokonce 4krát rychlejší. Zvláště u posledního pluginu je zjištění překvapující, protože rozdíl v měřeních je přibližně 100 milisekund.




Obrázek 7.4: Test zápisu dat do zařízení

## 7.6 Výsledek testů čtení dat ze zařízení

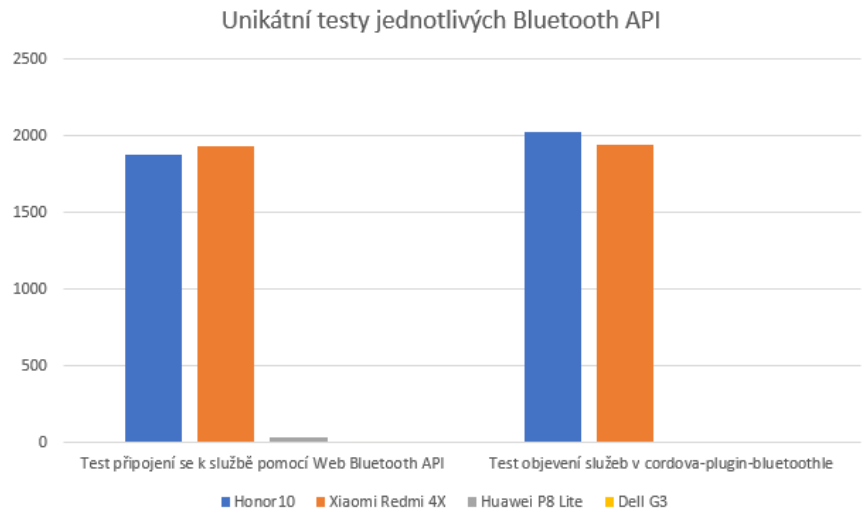
Již pátým testem byl test čtení dat z periferního zařízení. Výsledky testu můžeme vidět na Obrázku 7.5. Celkově jsem naměřil hodnoty okolo 100 milisekund. Největší rozdíl můžeme sledovat u Web Bluetooth API, kde zařízení značky Xiaomi vyčítalo data během desítek milisekund. Další 2 zařízení se přitom pohybovala okolo 100 milisekund. U následujících dvou pluginů vyšly podobné výsledky. výkonnější zařízení bylo vždy lehce pomalejší.




Obrázek 7.5: Test čtení dat ze zařízení

## 7.7 Výsledek testů připojení se k službě a zjištění dostupných služeb

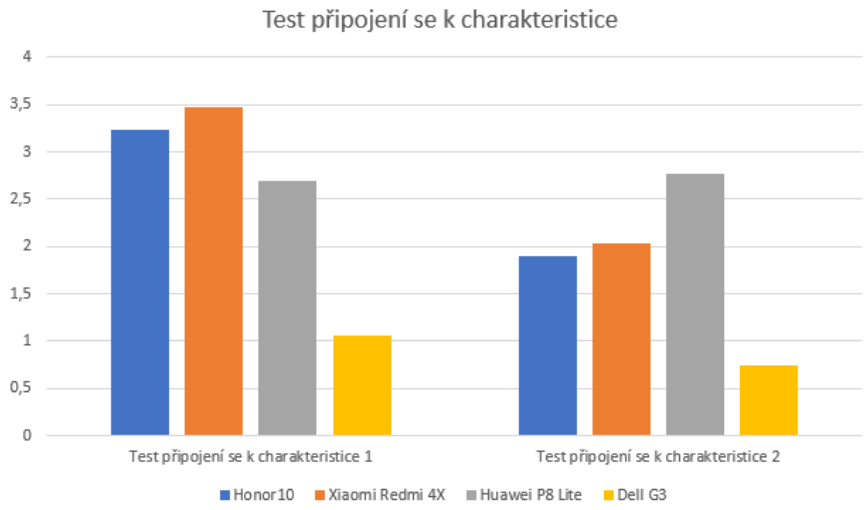
Zde jsem se v jedné části grafu zabýval připojením se k službě pomocí Web Bluetooth API. V druhé polovině grafu můžeme vidět naměřené hodnoty zjištění dostupných služeb pomocí pluginu cordova-plugin-bluetoothle. Grafy těchto částí najdeme na Obrázku 7.6. Oba výsledky se pohybují v řádech sekund s tím, že výkonnější zařízení bylo nepatrně rychlejší. Při připojování se k službě můžeme pozorovat extrémně rychlé provedení měření u zařízení značky Huawei (36 milisekund) a ještě rychlejší u zařízení Dell G3 15 Gaming, kde jsme dosáhli 4 milisekund. Po zjištění dostupných služeb bylo rychlejší méně výkonné zařízení, ale jen přibližně o 100 milisekund.




Obrázek 7.6: Test připojení se k službě a objevení služeb

## 7.8 Výsledek testů připojení se k charakteristice

Předmětem bylo provedení posledních dvou testů, které se nacházejí na Obrázku 7.7, kde jsem měřil přístup do charakteristiky před zápisem a před čtením naměřených hodnot. Charakterisktika 1 byla pro zápis a charakteristika 2 pro čtení. Pohybujeme se zde v jednotkách milisekund. V první charakteristice bylo nejrychlejším zařízením Dell G3 15 Gaming a nejrychlejší mobilní zařízení bylo Huawei P8 Lite. Následoval Honor 10, a poslední se umístilo Xaiomi. Byly zde rozdíly jen v milisekundách. U druhé charakteristiky bylo zařízení Dell G3 15 Gaming, Honor 10 a Xiaomi rychlejší než v prvním testu. Zařízení Huawei naměřilo téměř stejný výsledek. Nejrychlejší zde byl Dell G3 Gaming, následoval Honor a v těsném závěsu s dobrým časem provedení bylo Xiaomi.




Obrázek 7.7: Test připojení se k charakteristice

# Kapitola 8

# Závěr

V bakalářské práci jsme úspěšně naimplementovali a provedli testy pro jedny z nejužívanějších Bluetooth API. Testy zjistily, že nejvhodnějším Bluetooth API pro zjištění dostupnosti periferií je Web Bluetooth API. Čas, za který se kontrola provede je opravdu malý v porovnání s Cordova pluginy. Pokud bychom měli vyhledávat velké množství dostupných zařízení, bude nejlepší zvolit cordova-plugin-bluetoothle. Na obou testovaných zařízeních vyhledání proběhlo velmi rychle. Zároveň jsme zjistili, že pro navázání spojení se nehodí plugin cordova-plugin-central. Jeho testovací časy jsou mnohonásobně delší než u konkurenčních Bluetooth API. Pro zápis dat do zařízení bych využil Web Bluetooth API, popřípadě by se dal využít plugin cordova-plugin-bluetoothle. Co se týče čtení hodnot ze zařízení, tak neuděláme ani s jedním Bluetooth API chybu. Všechny naměřené časy byly velmi podobné a rozdíly minimální.

Bohužel nebylo možné provést testy na platformě iOS, protože jsem neměl k dispozici ani počítač s MacOS pro překlad, ani zařízení s iOS pro testování. U prvního pluginu, cordova-plugin-central by neměl být problém aplikaci rovnou využít k testování. U druhého pluginu, cordova-plugin-bluetoothle by bylo potřeba změnit postup při zjišťování dostupných služeb a následně bychom museli přidat implementaci pro zjištění dostupných charakteristik.

Po vyhodnocených testech bych se při případné implementaci rozhodl, buď pro použití Web Bluetooth API z důvodu jednoduchého užívaní na zařízeních nebo bych zvolil plugin cordova-plugin-bluetoothle, kde i přes pracnější implementaci v kódu a rozsáhlejší dokumentaci je rychlost provádění metod lepší.

U některých zařízení mohly být výsledky ovlivněné implementací Bluetooth na čipset jeho výrobcem, přičemž se ukazují dlouhodobé zkušenosti firmy Qualcomm, kde i starší model předčí novější (a výkonnější) modely jiných výrobců s novější verzí Bluetooth. Ale abychom měli jistotu, že je to skutečně dáno výrobcem, museli bychom testy spustit na novějším Qualcomm čipset, aby se vyloučil vliv verze Bluetooth.

Je zde prostor pro rozšíření testů. Mohli bychom se zaměřit na komunikaci přes sériový port pomocí Bluetooth Classic. Posoudili bychom, jaké jsou rozdíly v samotných Bluetooth API a ná-

sledně by proběhlo porovnání rychlostí mezi Bluetooth Low Energy a Bluetooth Classic. Dále by bylo zajímavé naimplementovat testy, kde bychom přenášeli větší množství dat a sledovali, jak se nám liší velikost přenesených dat za určitý časový úsek. Souběžně by bylo vhodné sledovat vývoj Web Bluetooth API a při významnějších updatech provést rešerši a nové otestování funkcí.

Při zpracování bakalářské práce jsem měl možnost naučit se pracovat s Bluetooth technologií, s kterou jsem se dříve setkával pouze jako běžný uživatel. Zároveň jsem si procvičil programování v jazyce JavaScript. Přínosné pro mě bylo praktické využití pluginů Apache Cordova počínaje jejich instalací přes debugování pomocí konzole až po závěrečné otestování vytvořené aplikace a potvrzení její funkčnosti na periferních zařízeních.

# Literatura


1. *Bluetooth® Low Energy (LE)* [online] [cit. 2021-04-26]. Dostupné z: https://www.bluetooth.com/learn-about-bluetooth/radio-versions/.
2. WOOLLEY, Martin. *Příručka pro vývojáře Bluetooth* [online]. 2016 [cit. 2021-04-27]. Dostupné z: https://www.bluetooth.com/blog/a-developers-guide-to-bluetooth/.
3. *Understanding Bluetooth Profiles* [online] [cit. 2021-04-26]. Dostupné z: https://www.dignited.com/28387/understanding-bluetooth-profiles/.
4. *Bluetooth profiles* [online] [cit. 2021-04-26]. Dostupné z: https://www.electronics-notes.com/articles/connectivity/bluetooth/profiles.php.
5. *GATT | Introduction to Bluetooth* [online]. 2014 [cit. 2021-04-26]. Dostupné z: https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gatt.
6. *GAP | Introduction to Bluetooth* [online]. 2014 [cit. 2021-04-26]. Dostupné z: https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gap.
7. *JavaScript Versions* [online] [cit. 2021-04-26]. Dostupné z: https://www.w3schools.com/js/js_versions.asp.
8. *CC2650STK* [online]. 2015 [cit. 2021-04-27]. Dostupné z: https://www.ti.com/tool/CC2650STK?keyMatch=CC2650STK&tisearch=search-everything.
9. *Connect sensors to the cloud fast with TI's new SimpleLink SensorTag* [online]. 2015 [cit. 2021-04-26]. Dostupné z: https://www.microwavejournal.com/articles/24506-connect-sensors-to-the-cloud-fast-with-tis-new-simplelink-sensortag.
10. *APPB1.03 Connecting SGW8130 BLE Sensor Tag with External Gateways* [online] [cit. 2021-04-29]. Dostupné z: https://www.sgwireless.com/.
11. *Hybridní, nativní nebo webové aplikace?* [Online] [cit. 2021-04-26]. Dostupné z: http://www.hybridniaplikace.cz/srovnani.html.
12. RITA, Inês. *NATIVE VS. HYBRID VS. PWA: THE PROS AND CONS* [online]. 2020 [cit. 2021-04-26]. Dostupné z: https://www.imaginarycloud.com/blog/native-vs-hybrid-vs-pwa-the-pros-and-cons/.

13. *Web Bluetooth API* [online]. 2021 [cit. 2021-04-26]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Web_Bluetooth_API.

14. *Web Bluetooth API, Browser compatibility* [online]. 2021 [cit. 2021-04-26]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Web_Bluetooth_API.

15. DON. *Bluetooth Serial Plugin for PhoneGap* [online]. 2017 [cit. 2021-04-27]. Dostupné z: https://www.npmjs.com/package/cordova-plugin-bluetooth-serial.

16. WITEK122. *Bluetooth Serial Plugin for PhoneGap* [online]. 2016 [cit. 2021-04-27]. Dostupné z: https://www.npmjs.com/package/cordova-plugin-bluetooth-serial2.

17. KIWISOUP. *Bluetooth Classic Serial Plugin for Cordova* [online]. 2016 [cit. 2021-04-27]. Dostupné z: https://www.npmjs.com/package/cordova-plugin-bluetoothClassic-serial.

18. DON. *Bluetooth Low Energy (BLE) Central Plugin for Apache Cordova* [online]. 2020 [cit. 2021-04-27]. Dostupné z: https://www.npmjs.com/package/cordova-plugin-ble-central.

19. RANDDUSING. *Cordova Bluetooth LE Plugin* [online]. 2021 [cit. 2021-04-27]. Dostupné z: https://www.npmjs.com/package/cordova-plugin-bluetoothle.

20. EVOTHINGS. *Cordova BLE Plugin* [online]. 2016 [cit. 2021-04-27]. Dostupné z: https://www.npmjs.com/package/cordova-plugin-ble.

21. *Performance* [online]. 2021 [cit. 2021-04-30]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Performance.

22. YIHE, Wang. *BLE communication and data storage flow chart* [online]. 2016 [cit. 2021-04-28]. Dostupné z: https://www.researchgate.net/figure/BLE-communication-and-data-storage-flow-chart_fig5_312020357.

23. *Promise* [online]. 2021 [cit. 2021-04-27]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise.

24. *Herní notebook Dell G3 15* [online] [cit. 2021-04-28]. Dostupné z: https://www.dell.com/cz/p/g-series-15-3579-laptop/pd.

25. ŠVEC, Vašek. *Huawei P8 Lite: Klenot ze střední třídy dokořeněný příznivou cenou (recenze)* [online]. 2017 [cit. 2021-04-28]. Dostupné z: https://www.svetandroida.cz/huawei-p8-lite-recenze/.

26. *Xiaomi Redmi 4X* [online] [cit. 2021-04-28]. Dostupné z: https://smartmania.cz/katalog/xiaomi-redmi-4x/.

27. *Kirin 970 - HiSilicon* [online]. 2020 [cit. 2021-04-28]. Dostupné z: https://en.wikichip.org/wiki/hisilicon/kirin/970.

# Příloha A

# Seznam příloh

V přiloženém ZIP archivu nalezneme následující adresáře:

- **/CordovaApk** – instalační balíčky pro platformu Android
- **/WebBluetoothAPI** – řešení implementace testu pomocí Web Bluetooth API
- **/cordova-plugin-bluetoothle** – řešení implementace testu pomocí pluginu cordova-plugin-bluetoothle
- **/cordova-plugin-central** – řešení implementace testu pomocí pluginu cordova-plugin-central
- **/Výsledky** – tabulky ve formátu XLSX s detailními výsledky jednotlivých testů